CMPSC-265
Data Structures and Algorithms

Zaihan Yang
zyang13@suffolk.edu

Department of Math and Computer Science
Suffolk University

Fall 2019

Notice

- Midterm_Exam1 will be held on:
  - Time: Oct 23rd (Wednesday)
  - Location: the same classroom
  - Topics:
    - The 1st week to this week's topics
  - Closed-book and closed-notes

Midterm_Exam1 Review

- Time complexity analysis and Big-O notation
- Data structures: how data are organized
  - Array
  - Sorted array (ordered array)
  - Stack
  - Queue
  - Priority Queue
  - Linked list
- Algorithms: how to work on those data structures
  - Binary search
  - Sorting Algorithms: selection/Insertion/BubbleSort
  - merge sort and Quick sort

Time Complexity and Big-O Notation

- We use time complexity to mathematically model time efficiency.
- Complexity analysis gives us a function that maps from input size to the number of steps the algorithm takes.
- Big O notation defines the complexity order. Formally, we say a function g(n) is O(f(n)) (read: g(n) is Big O of f(n)) if and only if there is a positive integer $n_0 \in N$ and a constant $c > 0$ such that for all $n \geq n_0$, we have that $g(n) \leq c * f(n)$. Namely, f(n) therefore determines the asymptotic upper-bounds of the growth of g(n)
- The common complexity classes in order are:
  - $O(1)<O(\log n)<O(n)<O(n\log n)<O(n^2)<O(n^3)<O(2^n)$
  - In terms of their order-of-growth from slowest to fastest

Unsorted Array

- Properties:
  - Fixed length
  - Allocated with contiguous memory.
- Operations:
  - Insert data / delete data / swap data / Search for a data
- Number of comparisons and moves

	Insert	Linear Search	Delete
# of Moves	O(1)	0	O(n/2)=O(n)
# of Comparisons	0	O(n/2)=O(n)	O(n/2)=O(n)

- Pros: Efficient to access to each element
- Cons: Not efficient to delete

sorted Array

- Properties:
  - A general array but all elements are arranged in ascending (descending) order
- Operations:
  - Insert data / delete data / swap data / Search for a data
- Number of comparisons and moves

	Insert	Linear Search	Binary Search	Delete
# of Moves	O(1)	0	0	O(n/2)=O(n)
# of Comparisons	O(n)	O(n/2)=O(n)	O(logn)	O(n/2)=O(n)

- Pros: Efficient to access to each element by its associated index
- Cons: Not efficient to insert or delete.
- Can only apply the binary search on a sorted (ordered) array.

Stack

- Properties: LIFO (last-in-first-out) or FILO (first-in-last-out)
- Operations: push(add a new element); pop(delete the top element); peek (check the top element); isEmpty (whether empty).
- Implementation:
  - Use array
  - Use single-ended singly linked list
- In both implementation , the Efficiency:
- Stack Applications:
  - Method calls
  - Reverse string
  - Parenthesis matchin
  - Inflix/Postfix evaluation

Operations	Stack
push(enqueue)	O(1)
pop(dequeue)	O(1)
peek	O(1)
isEmpty	O(1)
isFull	O(1)
size	O(1)

Queue

- Properties: FIFO (first-in-first-out)
- Operations: enqueue(add a new element); dequeue(delete the top element); peek (check the top element); isEmpty (whether empty).
- Implementation:
  - Use array
  - Use double-ended singly linked list
- In both implementation , the Efficiency:
- Queue Applications:
  - Waiting in lines
  - Tree level-order traversal
  - …..

Operations	Queue
push(enqueue)	O(1)
pop(dequeue)	O(1)
peek	O(1)
isEmpty	O(1)
isFull	O(1)
size	O(1)

Priority Queue

- Properties: same as the general queue but each element in queue is associated with ‘priority. When dequeuing elements from the queue, the one with the highest priority will be dequeued (not their arrival order)
- Operations: enqueue(add a new element); dequeue(delete the top element); peek (check the top element); isEmpty (whether empty).
- Implementation:
  - Use sorted array
  - User sorted double-ended linked list
- In both implementation , the Efficiency:
- Queue Applications:
  - Operating System process scheduling
  - Graph traversal / processing
  - …..

Operations	Queue
push(enqueue)	O(N)
pop(dequeue)	O(1)
peek	O(1)
isEmpty	O(1)
isFull	O(1)
size	O(1)

Linked list

- Linked list is a composed of a list of nodes (links) , each node (link) holds data and contains a reference to the next node (link)
- Property:
  - Dynamic memory allocation (resizable)
  - No fixed capacity limit
  - No data locality (not contiguous memory allocation)
  - Efficient in insert or delete.
  - Not efficient in accessing each node
- Several types:
  - Single-ended singly linked list
  - Double-ended singly linked list
  - Double-ended doubly linked lilst
  - Sorted linked list

Linked List

- Basic Operations:
  - InsertFirst, insertLast, deleteFirst, deleteLast, Search, Delete any intermediate one, Traverse the linked list
  - No practical usage when applying binary search on sorted linked list

Operations	SSLL	DSLL	DDLL	Sorted SSLL
Insert at First	O(1)	O(1)	O(1)	O(N) for insertion
Insert at End	O(N)	O(1)	O(1)	
Delete from First	O(1)	O(1)	O(1)	O(1)
Delete from End	O(N)	O(N)	O(1)	O(N)
Traverse	O(N)	O(N)	O(N)	O(N)
Search for a specific node	O(N)	O(N)	O(N)	O(N)
Delete a specific node	O(N)	O(N)	O(N)	O(N)

Binary Search

- On a sorted array, Binary search compares the target value to the middle element of the array. If they are not equal, the half in which the target cannot lie is eliminated and the search continues on the remaining half, again taking the middle element to compare to the target value, and repeating this until the target value is found. If the search ends with the remaining half being empty, the target is not in the array.
- Can be implemented in both iterative-way and recursive-way.
- Worst-case: O(logN)
- Average-case: O(logN)
- Best-case: O(1)

Sorting Algorithm

- Elementary Sorting:
  - Selection; Insertion; Bubble
- More advanced:
  - Merge sort; Quick sort
  - Both apply the divide-and-conquer paradigm, and are often implemented using recursion
  - Merge sort: recursively divide the array into two even halves, and sort each of them. Merge two sorted sub-arrays.
  - Quick sort: Given a chosen pivot, recursively partition the array into two sub-arrays, where all numbers in the left sub-array are smaller than the pivot, while numbers in the right sub-array are greater than the pivot.
  - Choosing the pivot: always choose the first one, the last one,
  - The median of three elements (the first, the last, and the middle one)

Sorting Algorithm

Algorithm	Best Case	Average Case	Worst Case	In-Place?	Stable?
Selection	$O(N^2)$	$O(N^2)$	$O(N^2)$	Yes	No
Insertion	O(N)	$O(N^2)$	$O(N^2)$	Yes	Yes
Bubble	$O(N^2)$	$O(N^2)$	$O(N^2)$	Yes	Yes
Merge	O(NlogN)	O(NlogN)	O(NlogN)	No	Yes
Quick	O(NlogN)	O(NlogN)	$O(N^2)$	Yes	No

Recursion

- Iterative approach
  - Use loops to repeat a task, “while”, “for”, …
- Recursive approach
  - A method calls itself
  - Each time with different parameters
  - Until we reach a trivial base case

Several Recursion Problems

- Triangular numbers
- Factorial numbers
- Fibonacci numbers
- Binary Search
- All anagrams of a word
- Display the entire Linked list and others
- Towers of Hanoi
- Tree processing
- Graph processing